International Journal of Approximate Reasoning 53 (2012) 969–987

Contents lists available at SciVerse ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

Efficient indexing methods for recursive decompositions of Bayesian networks

Kevin Grant

Department of Math and Computer Science, University of Lethbridge, 4401 University Drive, Lethbridge, Alberta, Canada T1K 3M4

ARTICLE INFO

Article history:
Available online 9 April 2012

Keywords:
Bayesian networks
Conditioning graphs

ABSTRACT

We consider efficient indexing methods for conditioning graphs, which are a form of recursive decomposition for Bayesian networks. We compare two well-known methods for indexing, a *top-down* method and a *bottom-up* method, and discuss the redundancy that each of these suffer from. We present a new method for indexing that combines the advantages of each model in order to reduce this redundancy. We also introduce the concept of an *update manager*, which is a node in the conditioning graph that controls when other nodes update their current index. Empirical evaluations over a suite of standard test networks show a considerable reduction both in the amount of indexing computation that takes place, and the overall runtime required by the query algorithm.

1. Introduction

Computing posterior probabilities from a Bayesian network (henceforth referred to as *inference*) is an NP-hard task [1]. However, modern algorithms are able to feasibly perform inference on many networks. The standard algorithms, such as Junction Tree Processing (JTP) and Variable Elimination (VE), can perform inference in time and space exponential on the induced width of the associated elimination ordering of the variables; this is the best complexity for inference known to date [2,3].

Within this asymptotic complexity lies some performance variation. For example, the probability distributions of Bayesian networks are often implemented as linear arrays. Given this representation, the inference algorithm must be able to compute the array index that corresponds to a given context of the variables defining the distribution (henceforth referred to as *indexing*). Using a straightforward indexing approach, it can be shown that for inference operations such as multiplication and marginalization, the number of arithmetic operations required to index the arrays far exceeds the number of operations that are performed on the probabilities in the arrays. Several algorithms for efficient indexing have been presented for the standard inference methods such as JTP and VE [4,5], which reduce the computation required for indexing during the course of inference. While these methods work well for standard inference algorithms, it is not clear how these methods translate to conditioning methods, in particular, recursive conditioning models [6,7]. These recursive algorithms do not employ multiplication and marginalization in the same way as the standard elimination algorithms do, and therefore their indexing requirements can differ considerably.

In this paper, we consider efficient methods for indexing distributions in recursive conditioning models. We focus on conditioning graphs [7], as the indexing structure for these data structures is explicit, rather than implied. The main contribution of this paper is a new bidirectional indexing model for conditioning graphs that combines the advantages of two known indexing methods. We also introduce the concept of an *update manager*, which is a node in the Bayesian network that controls when other nodes update their current index. Evaluating this new approach over a set of standard benchmark networks, we observe a reduction in the number of indexing operations by at least 58% compared to current methods, and a corresponding time reduction of 32% or more. While this paper focuses on conditioning graphs, the discussed methods

E-mail address: kevin.grant2@uleth.ca kevin@cs.uleth.ca

0888-613X/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ijar.2012.03.001

should be applicable to other recursive conditioning algorithms [6,8], where efficient indexing methods have not been formally considered.

2. Background and previous work

We denote random variables with capital letters (e.g., X, Y, Z), and sets of variables with boldfaced capital letters $\mathbf{X} = \{X_1, \ldots, X_n\}$. Each random variable V has a discrete domain of finite size $|V|$ containing values $\mathcal{D}(V) = \{0, \ldots, |V| - 1\}$. An instantiation of a variable is denoted $V{=}v$, or v for short. A *context*, or instantiation of a set of variables, is denoted $\mathbf{X}{=}\mathbf{x}$ or simply $\mathbf{x}$.

Given a set of random variables $\mathbf{V} = \{V_1, \ldots, V_n\}$ with domain function $\mathcal{D}$, a Bayesian network is a tuple $\langle \mathbf{G}, \Phi \rangle$. $\mathbf{G}$ is an acyclic directed graph over the variables in $\mathbf{V}$. $\Phi = \{\phi_{V_1}, \ldots, \phi_{V_n}\}$ is a set of probability distributions with a one-to-one correspondence with the elements of $\mathbf{V}$. More specifically, each $\phi_{V_i} \in \Phi$ is the conditional probability of V_i given its parents in $\mathbf{G}$ (called *conditional probability tables* or CPTs). That is, if π_{V_i} represents the parents of V_i in $\mathbf{G}$, then $\phi_{V_i} = P(V_i|\pi_{V_i})$. The *definition* of a discrete variable function is the set of variables over which it is defined. Fig. 1 shows the graph of a Bayesian network, taken from [2].

An *elimination tree* (etree) [7] is a tree whose leaves and internal nodes correspond to the CPTs and variables of a Bayesian network, respectively. The structure of an etree is characterized as follows: all CPTs in the Bayesian network containing variable V_i in their definition are contained in the subtree of the node labeled with V_i. This characterization does not uniquely define an etree; Fig. 2 shows one of the possible etrees for the Bayesian network of Fig. 1. There are several techniques to build etrees for Bayesian networks [9], but we will not discuss them here.

The computation of posterior probabilities can be expressed using the following identity. Suppose $\mathbf{C} = \mathbf{c}$ is an arbitrary context:

$$P(X = x|\mathbf{C} = \mathbf{c}) = \frac{P(X = x \wedge \mathbf{C} = \mathbf{c})}{P(\mathbf{C} = \mathbf{c})} = \alpha P(X = x \wedge \mathbf{C} = \mathbf{c}),$$

where $\alpha = P(\mathbf{C} = \mathbf{c})^{-1}$ is a normalization constant. Thus the task of computing a posterior probability (e.g., $P(X = x|\mathbf{C} = \mathbf{c})$) can be accomplished by computing the probability of an extended context $P(X = x \wedge \mathbf{C} = \mathbf{c})$; henceforth the task of computing the probability of a given context will be the main focus of our algorithms. This calculation will require marginalization over any hidden variables, that is, variables that occur in the Bayesian network, but are not part of the context.

The function $\mathcal{P}$ (Fig. 3) is used to compute the probability of a given context over a subset of variables in that etree (henceforth referred to as a *query*). The algorithm takes two parameters as input: an etree node (T) and a context (c). The return type of the function is a non-negative floating-point value, whose semantics depend on the node on which the call is made. When the algorithm is called given a leaf node, the return value is the conditional probability consistent with the context c taken from the CPT stored in the leaf. When the algorithm is called on the root node, the return value is the unconditional probability of the given context. For all other nodes, the result is an intermediate value that is simply a non-negative floating-point value representing a factor or term in the calculation (similar to the values from any intermediate potentials in the VE algorithm [3]). Lines 4–6 in the algorithm handle the case in which the etree node appears in the given context by computing the product of the probabilities of the node's children. Lines 8–13 handle the case in which the etree node is to be marginalized, which is accomplished by explicit conditionalization and summation. To condition over a variable, each value from the variable's domain is assigned once, and that assignment is added to the current context, which is then passed down through the recursive calls to that node's children. Line 2 handles the base case, in which a leaf node is reached, by retrieving a value from the CPT using the given context. The following notation is used in the algorithm: if T is a leaf node, then ϕ_T represents the CPT at T. If T is an internal node, then V_T represents the variable labelling T, and ch_T represents its children. Further details regarding this algorithm can be found in [7].

The time complexity of $\mathcal{P}$ over an etree T is exponential on T's height, while the space complexity is exponential only on the largest family in the Bayesian network [7]. However, the time complexity can be improved through a technique

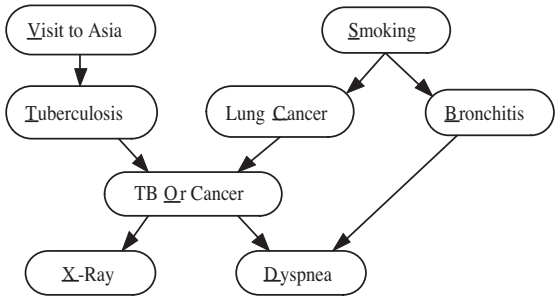

Fig. 1. An example Bayesian network.